Handreichungen des Kompetenzzentrums für MultiMedia und Telematik

Thema: XML – Workshop

Dr. B. Stumpp, M. Knobloch
KMMT am DIFF
Konrad Adenauer Str. 40
72072 Tübingen

1 Einführung

Am besten beginnen wir gleich mit einer kleinen Enttäuschung. Das vorliegende Skript ist Begleitmaterial zum Workshop XML, 9. Juni 1999 am KMMT im DIFF (Workshop-Betreuer: M. Knobloch, B. Stumpp). Es ist keine vollständige Übersicht über die XML-Sprache oder gar über die XSL-Style Sheets. Man sollte es eher als Erfahrungsbericht mit einem neuen, noch unvollständigen Standard begreifen, der die Autoren in einige tiefe Krisen gestürzt hat und manchmal erstaunliche Überraschungen mit sich brachte, kurz, wir geben an dieser Stelle eine Art Erfahrungsbericht wider und bieten eine Auswahl an konkreten Anwendungsbeispielen für XML und XSL an. Als Ergänzung und zum eigenen Weiterarbeiten haben wir Ihnen jede Menge Links und Literatur zusammengestellt.

Die Themen, die wir im Workshop behandeln wollen, sind folgende:

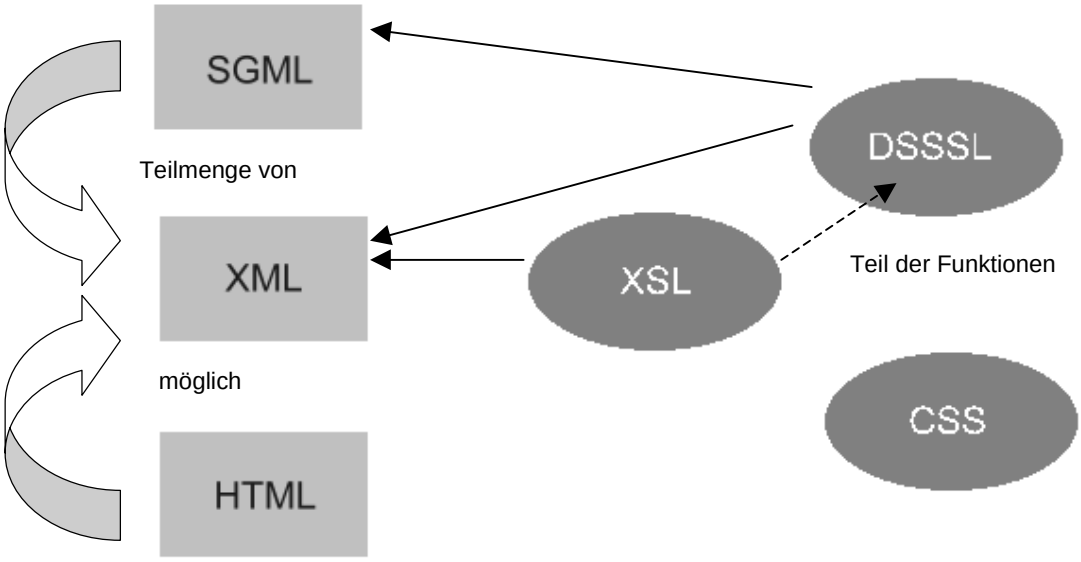
- Einführung in XML
- Praxisbericht: Elektronische Publikation mit SGML/XML
- Serverseitige Verarbeitung von XML/XSL mit dem Java-Servlet Cocoon
- XML-Anwendungsentwicklung mit eXcelon, dem Data Server von Object Design
- Einführung in RDF (Ressource Description Framework): Beschreibung und Austausch von Metadaten im Web

2 Eine kurze Geschichte von XML

XML hat als sogenannte „Markup-Sprache“ eigentlich zwei Vorgänger. Es ist aus der Standard Generalized Markup Language (SGML) entstanden. SGML ist ein internationaler Standard (ISO 8879) seit 1986. SGML beschreibt genau wie XML – welches eine Teilmenge von SGML ist – ein Format, wie man beschreibendes Markup („Textauszeichnungen“) in einem Dokument anbringen und damit eine logische Struktur in diesem Dokument aufbauen kann. SGML wird besonders auf dem Sektor der Verteidigung, in Industriezweigen wie der Luftfahrt, der Telekommunikation oder anderen Bereichen mit intensiver Dokumentation benutzt, ist aber nicht darauf beschränkt (Einführung s. http://www.arbortext.com/Think_Tank/SGML_Resources/Getting_Started_with_SGML/getting_started_with_sgml.html). Es braucht immer eine DTD, also ein Regularium für die Syntax. Ebenso braucht man zur Ausgabe besondere Stylesheets, die der DSSSL (Document Style Semantics and Specification Language) – Spezifikation folgen. SGML ist äußerst flexibel, aber auch komplex, teuer (i.e die Software) und nicht leicht zu erlernen.
Die zweite Wurzel von XML ist HTML, welches eine SGML-Anwendung ist. HTML wurde 1990 von Tim Berners Lee am Cern entwickelt. Als in den frühen 90er Jahren der erste grafische Browser Mosaic am NCSA entstand und das World Wide Web seinen beispiellosen Siegeszug begann, wurde HTML sozusagen über Nacht berühmt. Aber leider gelang nicht, seine Unverfälschtheit als SGML-Anwendung aufrecht zu erhalten, weil jeder ein Stückchen vom Kuchen haben wollte und jeder seine eigenen Lösungen erfand. XML soll unter anderem diese Probleme lösen helfen.

1996 gründet Jon Bosak von Sun Microsystems eine vor allem von den Entwicklern von SGML besetzte Arbeitsgruppe am W3C, welche eine Art SGML Lite entwickeln soll. Es soll die Stärken von SGML für das Web nutzbar machen, aber möglichst die Einfachheit von HTML beibehalten.
Tim Bray und Michael Sperberg-McQueen, Univ. Illinois, haben im wesentlichen die erste Spezifikation von XML verfaßt, welche im November 1996 publiziert wurde. Seither gibt es zahlreiche weitere Revisionen im Jahr. Parallel werden auch Standards für die mit XML verknüpften Technologien entwickelt, wie z.B. für bereits mehrfach erwähnte Style Sheet Sprache XSL, für Xlink bzw. XPointer (erweiterte Hyperlinks) etc. Details zu Autoren und Geschichte finden Sie unter der Überschrift „Overview“: http://www.oasis-open.org/cover/xml.html#reference

Die Skizze soll die Verbindungen zwischen diesen alten und neuen Standards veranschaulichen. Sie ist an das Schaubild aus Mintert/Behme, S. 17, angelehnt:



3 Was ist XML?

XML (Extensible Markup Language) ist eine standardisierte Metasprache, das bedeutet, mit dieser Sprache lassen sich wiederum beliebig viele Sprachen in der Art wie HTML es ist, beschreiben. Falls Sie also z.B. eine Genealogie verfassen wollten, würden Sie Personen, Geburts- und Sterbedaten, Familien, Ehen, Scheidungen etc. festlegen. Mit den sehr allgemein gehaltenen HTML-Tags wäre das fast unmöglich. Mit XML können Sie selbst Elemente definieren, die Sie für Ihr Dokument brauchen:

```
<Person Geschlecht= „m“>
<Geburt>
</Geburt>
<Ehe><Eheschliessung></Eheschliessung></Ehe>
...
</Person>
```

Die Elemente und Attribute, die Sie benutzen, müssen den in der Spezifikation von XML (s. Urls) festgelegten Regeln folgen, also z.B. muß jedes Element-Tag auch wieder geschlossen werden, also nicht wie bei HTML, welches Ausnahmen zuläßt, z.B. beim Element oder beim Element
. In XML sollte das .. bzw.
 heißen. Letzteres bedeutet, daß das Element-Tag geöffnet und dann wieder geschlossen wird. Die Regeln für Ihr Dokument können auch in einer separaten Datei oder einem separaten Dokumentteil abgelegt werden. Diese Dokumentation nennt sich DTD (Document Type Definition) und kann als Vokabular und Syntax für ihr Dokument, aber freilich auch für alle gleichartigen Dokumente wiederverwendet werden. Man kann daher fach- bzw. themenspezifische DTDs schreiben bzw. DTDs, die es schon gibt, als Vorlagen benutzen.

3.1 Was bedeutet Structured Markup Language?

XML beschreibt die Struktur und den Inhalt eines Dokumentes. Es sagt nichts über die Formatierung eines Elementes auf der Seite aus. Das steht in deutlichem Gegensatz zu HTML, welches Tags beinhaltet, in welchen Semantik, Struktur und Formatierung oft sogar miteinander verschmelzen. Wenn wir uns das HTML Element <H1> ansehen, dann erhalten wir eine Strukturierung als Überschrift der ersten Ebene und dazu noch eine bestimmte Formatierung, z.B. „Times, 24pt, fett“.
HTML-Elemente sind leider auch zu allgemein, so daß:
1. Suchmaschinen diese Daten nicht oder nur sehr unspezifisch auffinden können
2. die Inhalte nicht ohne Umwege in bzw. aus Datenbanken portiert werden können

Im folgenden Beispiel illustrieren wir diese Aussagen an einer Adressliste, die wir in ähnlicher Form als Basis für ein Expertennetz einsetzen:

HTML

```
...
<dt>Joachim Wedekind
    <dd>Adresse
        <ul>
            <li>Projekt: Projekt des Landes Ba-Wue
            <li>Institution: KMMT am DIFF
            <li>Strasse: Konrad-Adenauerstrasse 40
            <li>Ort: Tuebingen
...
        </ul>
    </dd>
</dt>
```

XML

```
...
<EXPERTE>
    <ADRESSE>
            <INSTITUTION TYP="Projekt des Landes BaWue"
            KOMMERZIELL="false">KMMT am DIFF</INSTITUTION>
            <STRASSE>Konrad Adenauerstrasse40</STRASSE>
            <ORT>Tuebingen</ORT>
...
</Adresse>
</Experte>
```

Sie können auf den ersten Blick sehen, daß die XML-Liste wesentlich besser strukturiert und lesbar ist. Die auszeichnenden Elemente sagen als solche schon etwas über ihren Inhalt aus, wohingegen ich bei HTML lediglich einen Listenpunkt erhalte. Die Art der semantischen Auszeichnung, wie sie bei XML erfolgt, macht eine Verarbeitung nicht nur für den Menschen, sondern auch für einen Rechner einfacher. Eine automatische Verarbeitung kann nicht erkennen, ob das Element <li> einen Namen, eine Adresse oder eine Anrede beinhaltet. Die Elementnamen von XML sind auf die Adressenverwaltung direkt zugeschnitten und könnten zum Beispiel gleichzeitig die Namensfelder in einer Datenbank sein.

3.2 Die Vorteile von XML auf einen Blick:

- *Dokumentenspezifisches Markup*
  Unterschiedliche Fachbereiche können Dokumente nach Vorstellungen generieren und diese austauschen
- *Allgemeines Datenformat*
  Schon heute sind viele ältere Datenformate (alte Word-, Word-Perfect oder Lotusdokumente etc.) nicht mehr oder ohne weiteres lesbar. XML ist reiner ASCII-Code und überdies vom W3C hervorragend dokumentiert.
- *Datenaustausch*
  Da XML nicht proprietär und einfach zu lesen und zu schreiben ist, eignet es sich sehr gut als Austauschformat zwischen Anwendungen
- *Dokumentenretrieval aus Datenbanken*
  XML-Daten lassen sich aufgrund ihrer guten Strukturierung in Datenbanken ablegen und aus diesen auch „on the fly“ generieren.
- *Metadaten*
  XML-Daten enthalten bereits Informationen über ihre Inhalte, die somit gut recherchierbar gemacht werden können

4 XML und Style Sheets (XSL, CSS)

Falls Sie Ihre Ergebnisse in einem Browser darstellen wollen, werden Sie sich jetzt zurecht fragen, wie denn Netscape oder IE in der Lage sein sollen, Tausende von Sprachen zu verstehen. Aber genau hierin liegt die Stärke von XML als Metasprache. Wenn eine Anwendung diese Metasprache versteht, versteht es auch automatisch alle Sprachen, die mit Hilfe dieser Metasprachen erstellt wurden.
Ein Browser findet nun die Tags, hat aber keine Möglichkeit, sie darzustellen, weil sie nicht wie die HTML-Tags samt ihrer Formatierungsanweisung fest in die Browser implementiert sind. Zur Wiedergabe bedarf es eines speziellen Style Sheets (XSL, möglich ist auch CSS), welches dem Dokument beigefügt wird. Auch das Style Sheet kann wie die DTD für beliebig viele Dokumente wiederverwendet werden (muß es auch, denn es steckt ziemlich viel Arbeit darin!).
Wir haben uns hier auf die Darstellung in einem Browser beschränkt, aber selbstverständlich kann für ein und dasselbe XML-Dokument auch ein Style Sheet für die Ausgabe auf einem Drucker oder einem anderen Medium geschrieben werden.

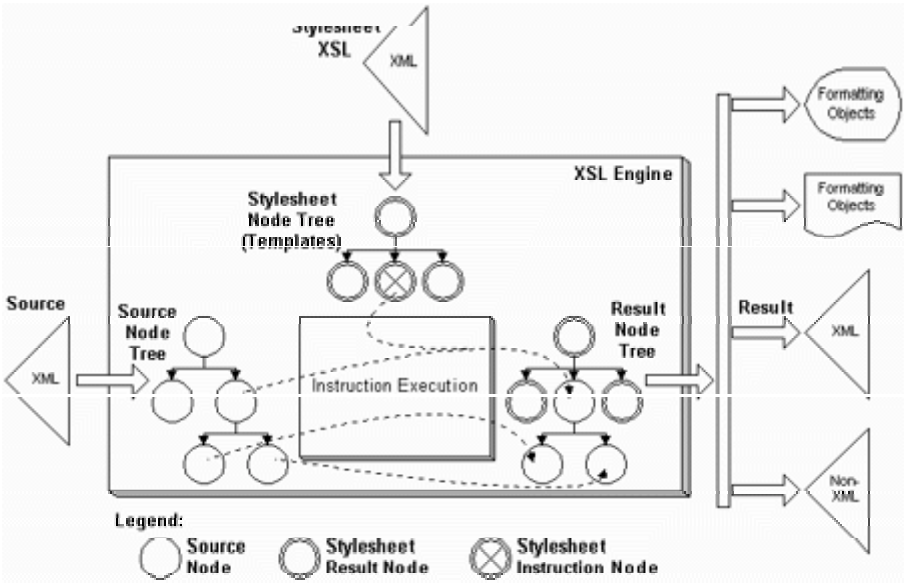
4.1 XSL – Aufbereitung eines XML Dokuments für die Darstellung

XSL (Extensible Style Language) hat Ähnlichkeiten mit zwei anderen ‚Formatierungssprachen', CSS und DSSSL, genauer gesagt, bietet einen Teil des Funktionsumfanges von DSSSL. DSSSL ist seit 1996 Standard, ist aber ziemlich diffizil und trotz der DSSSL-O (O wie online) nicht besonders webfreundlich (mehr dazu: http://www.jclark.com/dsssl). Während CSS (Cascadings Style Sheets) relativ einfach zu handhaben sind und bereits zur einheitlicheren und einfacheren Formatierung im Web benutzt werden, ist DSSSL eine ‚echte' Programmiersprache. Da XSL im Frühjahr 99 noch immer den Status eines ‚Working Draft' einnimmt, haben sich viele Autoren damit beholfen, ihre Dokumente mit CSS oder DSSSL bzw. DSSSL präsentationsfähig zu machen. Ein kleines Beispiel mit CSS präsentieren wir unten (s. S. 13).

Der gegenwärtige Umfang von XSL ist im Stadium eines Working Draft (21. 04. 1999, Beschreibung: http://www.w3.org/TR/WD-xslt), also kein fertiger Standard. Die XSL Sprache besteht aus 2 Teilen, die auch in zwei gesonderten Drafts zusammengefaßt sind:

1. Einer Sprache, um Dokumente zu transformieren
2. Einem XML-Vokabular, um Formatierungsanweisungen festzulegen

Abb. aus: K. Holman, Whats the Big Deal with XSL?, www.xml.com



4.2 Das XSL-Datenmodell

XSLT bezieht sich auf XML Dokumente, die als Baumstruktur repräsentiert werden. Die Verzweigungselemente des Baums (nodes) werden wie folgt unterschieden

- Wurzelknoten (root node), stellt den Einstiegspunkt in den Dokumentenbaum dar, d.h. das einzige Kind des root node ist das Dokument

- Elementknoten (element nodes). Ein Elementknoten können weitere Subelemente oder Attribute enthalten.
- Textknoten (text nodes) - Knoten verweisen auf keine weiteren Unterelemente, sondern lediglich auf Inhalt
- Attribute (attribute nodes)
- Elemente, die einen Namensraum festlegen (namespace nodes)
- Verarbeitungsanweisungen (processing instruction nodes)
- Kommentare (comment nodes)

Bei der Verarbeitung der Inhalte der Baumstruktur sucht der XSL Prozessor für jeden Knoten im Baum eine Verarbeitungsvorschrift. Diese Verarbeitungsvorschrift wird durch die template-rules (Vorlagen-Regeln) im XSL-Stylesheet festgelegt.

4.3 Wie arbeitet XSL?

Ein XSL Stylesheet enthält Vorlagen-Regeln (template rules). Diese haben in XSL immer zwei Teile:

1. Ein Muster, das den Knoten XML-Dokument identifiziert, auf den eine Anweisung angewendet werden soll.
2. Eine Anweisung, die den ausgewählten Knoten formatiert oder verarbeitet und als Ergebnis zurückgibt.

Beispiel:

```
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/TR/WD-xsl"
        xmlns:fo="http://www.w3.org/TR/WD-xsl">
         <xsl:template match="memo">
           <fo:block font-size="80%"
               indent-start="1.5cm"
               indent-end="1.5cm">
            <xsl:process-children/>
           </fo:block>
         </xsl:template>
</xsl:stylesheet>
```

Der XSL-Prozessor trifft auf den Knoten (wir sagen wohl häufiger ‚Element') *memo* und wendet dann eine Formatierung an, die das Element im Blocksatz, mit Schriftgröße 80% und um 1.5cm eingerückt darstellt. Dasselbe soll er mit den Kindelementen tun <xsl:process-children/>, sofern diese nicht eigene template rules (hier nicht dargestellt!) aufweisen. Jedes ‚template' im Style Sheet instantiiert einen kleinen Teil des Ergebnisbaumes. XSL strickt dann aus den Teilen einen kompletten Ergebnisbaum.

Hinweis! Wie man die richtige Syntax anwendet, um Suchmuster zu finden, können Sie im Microsoft Tutorial unter folgender URL üben:
http://msdn.microsoft.com/xml/xslguide/xsl-match.asp

Hinweis! Es ist wichtig zu wissen, daß die Transformierung vom Quellbaum in einen Ergebnisbaum auch ohne die Formatierungsanweisungen ablaufen kann. Standard-Transformationen konvertieren z.B. XML nach HTML. Im Moment, Frühjahr 1999,

unterstützen die meisten XSL Implementationen nur ersteres, nicht die Formatierungsobjekte!

Ein Element im XSL-Stylesheet ist entweder eine XSL-Processing Instruction (sie beginnt immer mit xsl:) oder ein Objekt, das in den Ergebnisbaum kopiert wird.
So wird im nachfolgenden Beispiel XSL benutzt, um XML in HTML umzuformen.
Jedes Element des memo-Dokumentes wird in den Ergebnisbaum kopiert, bis die Anweisung `<xsl:apply-templates />` kommt.
Beispiel:

```
<xsl:template match="memo">
   <HTML>
      <HEAD>
        <TITLE>Notiz</TITLE>
      </HEAD>
      <BODY>
        <xsl:apply-templates />
      </BODY>
    </HTML>
</xsl:template>
```

Sobald `<xsl:apply-templates />` an der Reihe ist, verarbeitet der XSL-Prozessor jedes der Kinder dieses Knotens. Er findet die angebenen Vorlagen und instantiiert sie. Die Abfolge dieser Vorlagen wird im Ergebnisbaum plaziert.

4.4 Anwenden von Vorlagenregeln (template rules)

Im obigen Beispiel führt die Anweisung `<xsl:apply-templates/>` dazu, daß nach weiteren Kindknoten des aktuellen Knotens gesucht wird. Wird `<xsl:apply-templates/>` ohne ein Attribut mit Namen *select* angegeben – also so wie hier -, werden alle Kindelemente des aktuellen Elements verarbeitet. Mithilfe des select Attributes kann aus der Menge der Kind-Elemente eine weitere Auswahl getroffen werden. Die Verarbeitung der Kindelemente geschieht, indem wieder nach template rules in der xsl-Datei gesucht wird, die auf die entsprechenden Elemente passen. Werden keine expliziten Regeln gefunden, werden die eingebauten Standardregeln angewendet (s. unten, S. Schablonen 11).
Ebenfalls ist es möglich Elemente zu verarbeiten, die keine Abkömmlinge des aktuellen Knotens sind. Das "Abteilung-Arbeitsgruppe-Angestellter" Beispiel zeigt dies:

```
<xsl:template match="angestellte">
  <fo:block>
    angestellte <xsl:apply-templates select="name"/> arbeitet in der
    <xsl:apply-templates select="from-ancestors(abteilung)/arbeitsgruppe"/>
     als Softwareentwicklerin.
  </fo:block>
</xsl:template>
```

Die mehrfache Verwendung von `<xsl:apply-templates/>` kann für einfache Umsortierungen bzw. Änderungen der Anordnung im Dokument benutzt werden. Nachfolgend werden zwei HTML-Tabellen erzeugt, die die Inlandsverkäufe von den Auslandsverkäufen getrennt auflisten. Die Reihenfolge im XML Dokument stellen wir uns entsprechend Zugangsreihenfolge vor.

```
<xsl:template match="produkt">
  <TABLE>
```

```
    <xsl:apply-templates select="verkauf/inland"/>
  </TABLE>
  <TABLE>
    <xsl:apply-templates select="verkauf/ausland"/>
  </TABLE>
</xsl:template>
```

XSL bietet zwar keine dynamischen Elemente wie JavaScript, kennt jedoch logische Konstrukte zur Ablaufsteuerung:

Wiederholung mit for-each

Das Beispiel gibt für alle Kindelemente des Elementes ARBEITSGEBIETE den Elementnamen und -inhalt in einer Tabelle aus:

```
<xsl:for-each select="./ARBEITSGEBIETE/*">
    <TR>
        <TD><xsl:node-name select="."/> </TD>
        <TD><b><xsl:value-of/></b> </TD>
        </TR>
    </xsl:for-each>
```

Bedingte Verarbeitung mit if (conditional processing)

Das `xsl:if` Element hat ein Attribut *test*, welches einen logischen Ausdruck definiert. Der Inhalt des Ausdrucks ist eine template rule. Evaluiert der Ausdruck zu *true*, wird der Inhalt des if-Blocks erzeugt, bei evaluation zu *false* wird der Block nicht ausgeführt.

Beispiel:

```
<xsl:template match="namelist/name">
  <xsl:apply-templates/>
  <xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

Zwischen den Namen, die mit apply-templates ausgegeben werden, gibt das Stylesheet ein Komma aus, es sei denn es handelt sich um den letzten Namen in der Liste.

Weitere Regeltypen und Kontrollstrukturen sind im XSLT Draft angesprochen, die hier nicht alle genannt werden sollen. Dieser kurze Einblick soll genügen, um einen prinzipiellen Ablauf der Verarbeitung des Dokumentenbaums im Folgenden genauer anzusehen.

4.5 Internet Explorer 5 und XML/XSL

Von den am Markt üblichen Browsern unterstützt derzeit lediglich der MS Internet Explorer 5 eine Untermenge des Transformationsteils des XSL- Working Draft vom 16. Dez. 1998.

Er kann daher:

- XML Dokumente direkt im Browser anzeigen, entweder mit einem eingebauten Default-XSL-StyleSheet, welches die Baumstruktur des Dokumentes anzeigt oder mit einem eigenen XSL-Stylesheet, welches z.B. HTML produziert (Link zum Default-Stylesheet: http://msdn.microsoft.com/workshop/c-frame.htm?927806185912#/workshop/xml/default.asp lokal: msxml.dll/DEFAULTSS.XSL)

- Er kann XSL-Transformationen auf dem Server durchführen, um HTML für nicht xml-fähige Browser zu generieren.

Probleme bzw. Bugs:

- Als *namespace* Angabe akzeptiert der IE 5 zur Deklaration von HTML nicht jedes beliebige Praefix, sondern nur **html**.
- Zeichen wie **<** kann nicht in XML-Textabschnitten dargestellt werden. Man muß die browserintegrierten Codierungen wie < (Codierungen s. Urls im Anhang) benutzen oder CDATA Sections anwenden (also z.B. den Ausdruck `<xsl:apply-templates />` so umhüllen: **`<![CDATA[`**`<xsl:apply-templates />`**`]]>`**

- Folgende im Working Draft festgelegte Schablonen sind im IE 5 **nicht** berücksichtigt, was zu großen Schwierigkeiten führen kann, wie das Beispiel unten (S. 18) zeigen wird:

  a) um ein rekursives Verarbeiten auch dann fortsetzen zu können, wenn es kein passendes Muster gibt, gibt es eine Regel, die für Elementknoten und Wurzelknoten gilt:

```
<xsl:template match="*|/">
<xsl:apply-templates/>
</xsl:template>
```

  b) Um auch Textknoten durchkopieren zu können, gibt es zusätzlich:

```
<xsl:template match="text()">
<xsl:value-of select="."/>
</xsl:template>
```

  Diese eingebauten Regeln werden so behandelt, als ob sie vor dem benutzerdefinierten Stylesheet importiert worden wären, d.h. sie werden als weniger wichtig angesehen als andere, benutzerdefinierte Vorlagenregeln.

Somit kann der Autor ein solche eingebaute Regel durch das explizite Angeben einer Regel mit match="*|/" or match="text()" überschreiben.

Selbstverständlich können Sie stattdessen auch einen der zahlreichen anderen XSL-Prozessoren installieren und mit diesen experimentieren. Unter xmlsoftware.com bzw. WDVL.internet.com finden Sie eine Menge Auswahl. Die meisten Tools sind noch nicht in Applikationen verfügbar, haben also keine grafische Benutzeroberfläche und sind deshalb von der Kommandozeile aus zu bedienen.

5 Praktische Anwendungsbeispiele

Ohne Sie entmutigen zu wollen: im Moment muß noch sehr viel daran gearbeitet werden, um XSL, also die Formatierungs- und Stilkomponente zu einem vernünftigen Standard zu verabschieden (von anderen XML-verwandten Technologien ganz abgesehen). Experimentierfreude muß man schon mitbringen, da aufgrund des mangelnden Standards gerade die XSL Prozessoren oft in Details voneinander abweichen und nicht nur das: Eingeschworenen Nutzern grafischer Oberflächen sei gleich gesagt: Die meisten Parser und Prozessoren werden im Moment von der guten alten Kommandozeile aus aufgerufen.

CSS und XSL – Kurzvergleich

Für einen Vergleich von XML und CSS lassen wir am besten Jon Bosak, den Vorsitzenden der XML Working Group beim W3C, selbst sprechen. Er hat in einem Vortrag aufgelistet, welche Beschränkungen es bei der Arbeit mit CSS gibt:

- CSS kann eine Einheit (etwa eine Kapitel-Überschrift) nicht auch an einem anderen Platz in dem Dokument erneut anzeigen (etwa in der Kopfzeile).

- CSS kann keine Zwillings-Beziehungen schaffen. Zum Beispiel ist es nicht möglich, eine Formatvorlage zu erstellen, die jeden zweiten Absatz im Fettdruck erscheinen läßt.

- CSS unterstützt keine IF-Schleifen; Erweiterungen durch den Ersteller der Formatvorlagen sind nicht möglich.

- CSS kann weder Einheiten durchzählen noch Werte in Variablen speichern. Das bedeutet, daß es nicht einmal möglich ist, gebräuchliche Parameter an einem Platz zu versammeln, an dem sie leicht verändert werden können.

- CSS kann nicht eigenständig Text erzeugen (etwa Seitenzahlen und dergleichen).

- CSS liefert ein simples kasten-orientiertes Formatierungs-Modell, das für die bisherigen Web-Browser gut geeignet ist, nicht aber für komplexere Aufgaben.

Diese und weitere Beschränkungen können mit XSL überwunden werden. Es mag nicht der Weisheit letzter Schluß sein, aber wer wirklich sofort Dokumente auf XML-Basis für die gängigen Browser anbieten und sich dabei einer clientseitigen Verarbeitung bedienen will, der kann XML mit CSS formatieren. Man hat immerhin den

Vorteil, die eigenen Dokumentinhalte mit sinnvollen und spezifischen Auszeichnungen versehen zu haben.

5.1 Anwendungsbeispiel 1: CSS und XML

Das folgende Anwendungsbeispiel ist so einfach wie möglich gehalten, weil wir nur auf die wichtigsten Details hinweisen wollen.
Im ersten Beispiel sind Stylesheet und XML-Dokument getrennt. Durch die Processing Instruction **`<?xml-stylesheet type="text/css" href="pre.css" ?>`** wird das xml-Dokument mit dem externen Style-Sheet verknüpft:

XML-Dokument *pre.xml:*

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="pre.css" ?>
<verfasser xmlns:HTML="http://www.w3.org/Profiles/XHTML-Transitional">
Karl Napf, der Verstimmte
<title>XML: Die neue Plage der Menschheit</title>
<bild><HTML:A href="http://www.xmlsoftware.com"><HTML:IMG
src="tort.gif"/>
</HTML:A>
</bild>
</verfasser>
```

XSL-Stylesheet *pre.css:*

```
verfasser
   {display:block; font-family: verdana, sans-serif; color: #888833; font-
    size: x-large; text-align: center;}
title
    {display:block; font-size: medium; padding: 3em;}
bild
    {display:block; width:250px; height:150px;}
HTML\:IMG
    {border: 5px solid black;}
HTML\:A:link  HTML\:IMG
    {border-color: blue;}
HTML\:A:visited  HTML\:IMG{border-color: grey;}
HTML\:A:active HTML\:IMG
    {border-color: red;}
```

Worauf Sie achten müssen:

1. Die *display* Eigenschaft von XML ist standardmäßig auf display:inline gesetzt. Sie müssen sie in den meisten Fällen auf display:block setzen (Genaueres finden Sie in den CSS-Anleitungen: http://www.teamone.de/selfhtml/tdch.htm#a18, http://www.w3.org/TR/REC-CSS1#display – auf CSS können wir leider nicht näher eingehen)

2. Namensräume (namespaces: z.B. **xmlns:HTML= „ *href* “** – näheres s. http://www.w3.org/TR/1999/REC-xml-names-19990114) müssen im Wurzelelement, das ist hier `<verfasser>` deklariert werden. Damit sind sie global gültig.

3. Man kann keine default namespaces angeben. Jedes namespace Element muß ein explizites Praefix haben. CSS Selektoren können ein Element eines Namensraumes nur durch das Praefix bestimmen, d.h. Sie müssen auf Konsistenz der Praefixe im XML Dokument und im CSS Sheet achten, weil Sie sonst dem Chaos Vorschub leisten.

4. Wir benutzen im Beispiel den HTML-Namensraum. Er kann Elemente genau wie in HTML darstellen. Das haben wir bei <A> und <IMG> benutzt, damit wir Bilder darstellen und in gewohnter Maier Links setzen können. **Achtung: Alle Tags müssen auch wieder geschlossen werden, also z.B.** `<IMG src="tort.gif"/>.`

5. Damit CSS den Namensraum von einer Pseudoklasse unterscheiden kann – beide haben einen Doppelpunkt in der Notation - muß der Namensraum HTML mit einem \ maskiert werden: z.B. HTML\:IMG

6. Im IE 5 muß der Namensraum für HTML auch *HTML* oder *html* heißen, sonst funktioniert gar nichts. Das widerspricht der Spezifikation eigentlich, da sie beliebig Prefixes erlaubt.

Anstelle des externen Stylesheets könnten die CSS Anweisungen auch direkt in das XML Dokument hineingeschrieben werden. Allerdings halte ich das nicht für die beste Lösung, weil Änderungen an einem zentral abgelegten Style Sheet sehr viel leichter zu warten sind.
Der Kopf des Dokumentes sähe dann so aus:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" ?>
<verfasser xmlns:HTML="http://www.w3.org/Profiles/XHTML-Transitional">
    <HTML:Style>
Stylesheet-Anweisungen
    </HTML:STYLE>
...
```

Heraus kommt folgende Seite: Ein Verfassername, ein Buchtitel und ein Bild (Comic von Glasbergen – mit leicht geänderter Bild-Unterschrift von mir):

5.2 Anwendungsbeispiel 2: Auswahl von Daten aus größeren XML-Beständen

Diese Beispiel ist als Einstieg ganz gut geeignet, weil wir im wesentlichen eine HTML Vorlage als Basis für ein XSL-Sheet beutzen. Für hochstrukturierte, sich immer wieder wiederholende Dokumentarten (Kataloge, Warenbestände, Adressdateien, Museumsbestände, Lexikaeinträge etc.) kann man zum Beispiel eine HTML-Tabelle als Vorlage nehmen und diese als XSL-Stylesheet aufbereiten.

Man kann dann über eine Stylesheet Anweisung die Daten, die im Browser angezeigt werden sollen, aus dem XML Dokument holen. Vorausgesetzt wird, daß Sie bereits über XML Daten verfügen.

Voraussetzung: IE 5 als Browser

Man kann in drei Schritten vorgehen:

1. Erstellen der HTML Vorlage
2. Aufbereiten der Vorlage als einfaches XSL-Stylesheet
3. Ausgabe der Daten in gewünschter Auswahl im Browser

XML-Dokument *experten.xml:*

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="templ01.xsl"?>
<!DOCTYPE EXPERTENNETZ SYSTEM "dress01.dtd">
<EXPERTENNETZ>
   <EXPERTE>
      <ADRESSE>
         <TITEL>St.g.Inf.</TITEL>
         <VORNAME>Manfred</VORNAME>
         <NAME>Knobloch</NAME>
           <INSTITUTION TYP="Projekt des Landes BaWue"
           KOMMERZIELL="false">KMMT am DIFF</INSTITUTION>
         <STRASSE>Konrad Adenauer Strasse 40</STRASSE>
         <ORT>Tuebingen</ORT>
         <PLZ>72072</PLZ>
         <LAND>Deutschland</LAND>
         <TELEFON>07071/979-232</TELEFON>
         <FAX>07071/979-322</FAX>
         <EMAIL>knobloch@kmmt.diff.uni-tuebingen.de</EMAIL>
         <URL>http://kmmt.diff.uni-tuebingen.de</URL>
         <FOTO/>
      </ADRESSE>
      <FACHGEBIET>
         <BEREICH>NETZE</BEREICH>
           <PROJEKT>relationale und objektorientierte Datenbanken, DB2,
           Datenbank des KMMT</PROJEKT>
         <WEBINFOS/>
         <PUBLIKATIONEN>&lt; XML&gt; ist léger</PUBLIKATIONEN>
      </FACHGEBIET>
    usw.   ...
```

1. Die HTML-Datei als Vorlage ist eine ganz schlichte Tabelle.
In diesem Beispiel werden wir aus der XML-Quelle lediglich drei Felder herausnehmen, nämlich NAME, EMAIL und INSTITUTION.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<Table Border="3">
<TR>
<TD>NAME</TD>
<TD>INSTITUTION</TD>
<TD>EMAIL</TD>
</TR>
<TR>
<TD><!— hier müßte der Name rein!-></TD>
<TD><!— hier müßte die Institution rein!- ></TD>
<TD><!— hier müßte die Email rein!-></TD>
</TR>
</Table>
</BODY>
</HTML>
```

2. Die Umformung der HTML-Vorlage in ein XSL-Stylesheet ist selbst eine XML Datei und braucht (alles Fettgedruckte haben wir hinzugefügt):

- eine gültige Deklaration **<?xml version='1.0'?>**

- einen gültigen Namensraum <xsl:stylesheet **xmlns:xsl="http://www.w3.org/TR/WD-xsl">**
- eine Processing Instruction<**xsl:stylesheet** xmlns:xsl="http://www.w3.org/TR/WD-xsl">
- eine xsl-Anweisung, um eine Vorlage auf multiple Knoten (z.B. für jedes Element Name) anzuwenden: **<xsl:for-each select="EXPERTENNETZ/EXPERTE/ADRESSE">**
- ein select-Attribut, das sagt, für welchen Knoten die Anweisung gelten soll, zB. **select="NAME"**
- xsl-Anweisungen, die den Wert des ausgewählten Knotens wiedergeben sollen, z.B. **xsl:value-of select="NAME"/>**

Alle Elemente müssen auch wieder geschlossen weden!

templ01.xsl:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<HEAD>
</HEAD>
<BODY>
<Table Border="3">
<TR>
<TD>NAME</TD>
<TD>INSTITUTION</TD>
<TD>EMAIL</TD>
</TR>
<xsl:for-each select="EXPERTENNETZ/EXPERTE/ADRESSE">
<TR>
<TD><xsl:value-of select="NAME"/></TD>
<TD><xsl:value-of select="INSTITUTION"/></TD>
<TD><xsl:value-of select="EMAIL"/></TD>
</TR>
</xsl:for-each>
</Table>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

ACHTUNG! Ich hatte zuerst ein leeres <Title> Tag in die Vorlage mitaufgenommen. Darauf hat IE 5 dann gar nichts angezeigt, weil er keine Vorlage dazu in den XML Daten fand. Also liess ich das weg.

---

3. Was rauskommt, ist optisch freilich nicht besonders berauschend. Aber die Optik kann dann über Formatierungsanweisungen (auch direkt in den HTML-Tags!) aufpoliert werden. Aber wir wollen noch einmal darauf hinweisen, daß es mit XSL in Zukunft sehr einfach möglich ist, Selektionen aus größeren XML-Datenbeständen vorzunehmen und diese direkt im Web darzustellen. Selbstverständlich können dann auch unterschiedliche views für unterschiedliche Benutzergruppen über XSL Sheets

realisiert werden:

NAME	INSTITUTION	EMAIL
Knobloch	KMMT am DIFF	knobloch@kmmt.diff.uni-tuebingen.de
Bett	KMMT	bett@kmmt.diff.uni-tuebingen.de
Wedekind	KMMT am DIFF	wedekind@uni-tuebingen.de

Das Beispiel kann erweitert werden, indem man nicht nur Elementwerte sondern auch Attributwerte auswählen läßt. Wählt man ein Attribut aus, so muß vor dem Attribut immer ein @ stehen, z.B. @TYP. Wir haben das obige Beispiel erweitert und zu den Elementen noch die Spalte mit der gleichnamigen Überschrift „TYP“ hinzugefügt.

templ02.xsl:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<HEAD>
</HEAD>
<BODY>
<Table BORDER=“3“>
<TR>
<TD>TYP</TD>
<TD>NAME</TD>
<TD>INSTITUTION</TD>
<TD>EMAIL</TD>
</TR>
<xsl:for-each select="EXPERTENNETZ/EXPERTE/ADRESSE">
<TR>
<TD><xsl:value-of select="INSTITUTION/@TYP" /></TD>
<TD><xsl:value-of select="NAME"/></TD>
<TD><xsl:value-of select="INSTITUTION"/></TD>
<TD><xsl:value-of select="EMAIL"/></TD>
</TR>
</xsl:for-each>
</Table>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

TYP	NAME	INSTITUTION	EMAIL
Projekt des Landes BaWue	Knobloch	KMMT am DIFF	knobloch@kmmt.diff.uni-tuebingen.de
Projekt des Landes BaWue	Bett	KMMT	bett@kmmt.diff.uni-tuebingen.de
Projekt des Landes BaWue	Wedekind	KMMT am DIFF	wedekind@uni-tuebingen.de

5.3 Anwendungsbeispiel 3: Problem mit dem Default-XSL-Stylesheet des IE 5

Stellen Sie sich vor, Sie wollten Unterlagen für eine Grammatik aufbereiten. Dabei sollten einzelne Satzteile wie Subjekt, Prädikat, Objekt etc. unterschiedlich ausgezeichnet werden (also extra Markup erhalten), der sonstige Text soll einfach ausgegeben werden. Das sollte ja wohl kein Problem sein – dachten wir auch. War es

aber. Der Text des Knotens *Satz*, den wir als Suchmuster angegeben hatten erschien nicht im IE, lediglich die Inhalte Kindelemente des Knotens. Wir saßen tagelang, bis wir in der mailing-Liste die Antwort bekamen, daß das oben bereits erwähnte
```
<xsl:template match="text()">
<xsl:value-of select="."/>
</xsl:template>
```
schlicht fehlte. Anbei das kleine Beispiel zum Probieren:

html2xml.xml:
```
<?xml version='1.0' encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="html2xml.xsl"?>
<Absatz>
<Satz>
Karl
    <Subjekt>Napf, <Apposition>alias kleiner Topf</Apposition></Subjekt>
 erfand
       <Objekt>den Napf</Objekt>
oder auch das N&#220;pfsche.
</Satz>
</Absatz>
```

html2xml.xsl in der **nicht** funktionierenden Variante:
```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <xsl:apply-templates select="Absatz/Satz" />
</xsl:template>
<xsl:template match="Satz">
  <html>
    <head>
    <title>Karl Napf</title>
    </head>
    <body>
      <xsl:apply-templates />
    </body>
  </html>
</xsl:template>
<xsl:template match="Subjekt">
  <b><xsl:value-of /></b>
</xsl:template>
<xsl:template match="Objekt">
  <i><xsl:value-of /></i>
</xsl:template>
</xsl:stylesheet>
```

* Das Element <*Apposition*> wurde zu Testzwecken absichtlich nicht aufgenommen!

Ergebnis: IE 5 gibt folgenden unvollständigen Satz aus:

Napf, alias kleiner Topf *den Napf*

also <Subjekt>, die Kinder von <Subjekt> und <Objekt>

html2xml.xsl in der funktionierenden Variante:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <xsl:apply-templates select="Absatz/Satz" />
</xsl:template>
<xsl:template match="Satz">
  <html>
    <head>
    <title>Karl Napf</title>
    </head>
    <body>
      <xsl:apply-templates />
    </body>
  </html>
</xsl:template>
<xsl:template match="Subjekt">
  <b><xsl:value-of /></b>
</xsl:template>
<xsl:template match="Objekt">
  <i><xsl:value-of /></i>
</xsl:template>
<xsl:template match="text()">
<xsl:value-of />
</xsl:template>
</xsl:stylesheet>
```

Ergebnis: IE 5 gibt folgenden vollständigen Satz aus:

Karl **Napf, alias kleiner Topf** erfand *den Napf* oder auch das Näpfsche.

* Der Umlaut muß codiert werden (Liste zum Nachschlagen s. Urls im Anhang).

6 Tools zur Verarbeitung/Anwendungsentwicklung von XML/XSL

6.1 Servlets

Ein Problem wird auch die stark motivierte XML-Gemeinschaft in nächster Zeit wohl nicht bewältigen können und das ist das clientseitige Verarbeiten der XML-Dokumente aufgrund mangelnder oder unterschiedlicher Implementierung. Alle Webentwickler schlagen sich heute damit herum, die Funktionen von HTML 4.0 wenigstens in den verbreiteten Browsern einigermaßen konsistent wirken zu lassen.
Wenn man serverseitig XML und XSL einsetzt, kann man dieses Problem relativ elegant umsegeln. Freilich ist die Zeit für die Installation der Tools und das Herumschlagen mit dem XSL Working Draft und seiner unterschiedlichen Implementierung ernsthaft in Rechnung zu stellen.

Kommerzielle wie nicht kommerzielle Anbieter arbeiten an javabasierten Lösungen, die – theoretisch! – den Vorteil haben, zwischen diversen Plattformen/Webservern etc. portabel zu sein (im Gegensatz zu proprietären Schnittstellen wie NS-API von Netscape und ISAPI von Microsoft). In der Regel handelt es sich um Servlets. Servlets sind Java-Objekte, die in einem WWW-Server ablaufen. Erhält der WWW-Server einen HTTP-Request, der vom Servlet bedient werden soll, ruft er je nach Bedarf

doGet() oder do(Post) auf. Servlets sind in letzter Zeit sehr beliebt geworden, weil sie im Gegensatz zu CGI und FastCGI als thread innerhalb des Server-Prozesses ablaufen und das (zeit-)aufwendige Erzeugen neuer Prozesse entfällt.

Wir haben mit mehr oder minder großem Erfolg einige Applikationen getestet und dabei auch ziemlich viel Frust erlebt. Das liegt wiederum an den generell unfertigen Produkten im Alpha- oder Betastadium. Andererseits ist es wirklich spannend, ‚live' mitzuerleben, wie eine netzbasierte Arbeitsgruppe eine solche Lösung entwickelt.
Für die aufgeführten Beispiele braucht man generell ein JDK (Java Development Kit), das Java Servlet Development Kit sowie eine Servlet Engine, außerdem die jeweils für die Produkte angegebenen XML Parser und XSL Prozessoren (unten genauer ausgeführt).

6.1.1 Cocoon Servlet

Beschreibung
Cocoon ist in der Lage, Webdokumente in prinzipiell jedem Format auszugeben. Derzeit wird fast ausschließlich damit experimentiert, automatisch HTML-Code zu erzeugen, indem mit Hilfe des XSL Prozessors und der XSL-Datei die xml-Dokumente nach HTML transformiert werden.

Die Entwicklung des Cocoon Servlets, das in Anspruch nimmt, ein Web-Publishing-Framework werden zu wollen, geschieht im Rahmen des Apache Projekts, genauer in der Java Abteilung des Projects. Das Java Apache Projekt hat bereits eine Java Servlet Engine (ApacheJServ früher: mod_jserv) entwickelt, die die Basis für verschiedene Servlet Entwicklungen im Apache-Umfeld darstellt. Eines der jüngsten ist Cocoon. Die Version 1.0 wurde von Projektleiter Stefano Mazocchi im März 1999 herausgegeben. Inzwischen ist Version 1.3.1 verfügbar, parallel wird an Version 2.0 gearbeitet. Da am Projekt momentan heftigst gerbeitet wird und selbst die Versionen im Zweiwochenrythmus erneuert werden, ist es am besten, sich immer wieder auf der Website zu informieren: http://java.apache.org. Dort findet man Installationshinweise, Dokus, FAQs und die mailing-list der ‚Working Dogs'.

Verarbeitungsablauf

Das Servlet ist eine Rahmenanwendung, die einen XML-Parser und einen XSL-Prozessor verbindet. Das Servlet wird nicht direkt aufgerufen, sondern über die Angabe einer URL, die eine XML-Datei beschreibt.

Beispiel: http://mein.server.de/test.xml

Ein Eintrag in der Konfiguration des Webservers bewirkt nun, daß vom Webserver das *Cocoon Servlet* aufgerufen wird. Das Programm öffnet die XML-Datei, baut mit dem eingebundenen Parser einen Dokumentbaum auf, liest die XSL-Datei, die in der XML-Quelle angegeben war, und produziert mit dem angebundenen XSL-Prozessor den Output. Das ist in derzeit eigentlich immer HTML, muß es aber nicht sein.
Beide Varianten sind mit kostenloser Software für Education erstellt.

Installation und Konfiguration

Das Paket Cocoon kann von http://java.apache.org geladen werden. Im Paket ist der von den Entwicklern bevorzugte XML-Parser und XSL-Prozessor enthalten. Cocoon erlaubt es aber auch, andere Parser oder Prozessoren einzubinden.
Derzeit werden openxml und xslp mitgeliefert. Die Leistungsfähigkeit dieser Tools bestimmt auch, inwieweit Cocoon den Anspruch, den Standard zu erfüllen, verwirklichen kann. Zum Betrieb des Servlets werden ein Webserver und eine Servlet-Engine benötigt.

A) NT 4.0 mit Servlet Engine JRun

Um es gleich vorwegzunehmen, mit folgender Konfiguration lief Cocoon 1.3.1 erst nach langem Probieren. Zunächst waren auch noch viele Bugs in den Cocoon Versionen 1.0-1.3, letztlich lag es aber doch am Mapping von Jrun
Das Mapping von JRun unterscheidet sich nämlich von der augenblicklichen Referenzimplementation und entspricht vermutlich der in wenigen
Tagen zur JavaONE (Java Entwicklerkonferenz 15. - 18. Juni) erwarteten. Was Standard ist, ist z.Z. nicht klar.
Die deshalb leicht abgeänderte funktionierende Version des Cocoon.jar können Sie auf Anfrage (per email) von uns erhalten.
Alle Komponenten (außer NT) sind frei erhältlich.

- NT 4.0
- JRun 2.3.1 (Servlet-Engine von http://www.livesoftware.com (für Education frei!)
- JRun Webserver (Teil des Paketes Jrun Pro)
- JRE 1.2.1 (Teil des Paketes Jrun Pro)
- Cocoon 1.3.1 (aktuelle Versionen der ebenfalls benötigten Prozessoren OpenXml und XSL:P sind dabei).

Vorbereitungen:
- Cocoon.jar von java.apache.org herunterladen und in ein Extra Verzeichnis auspacken
- Jrun 2.3.1 herunterladen und mit dem Installations-Assistenten installieren. **Achtung:** Bei der Auswahl der Webserver vorerst *keinen externen Webserver* angeben! Damit wird der Jrun Webserver gewählt; Standard-Port: 8000. **Achtung:** Die folgenden Installationsanweisungen gehen davon aus, dass das
- mit JRun mitgelieferte JRE benutzt wird. Sonst sind die Anweisungen
- entsprechend zu ändern.

Im Verzeichnis cocoon\bin befinden sich drei jar.files
openxml.jar
xslp.jar
Cocoon.jar

Kopieren Sie openxml.jar und xslp.jar nach *Jrun\jre\lib\ext*
Kopieren Sie Cocoon.jar nach *Jrun\servlets*
Kopieren Sie außerdem cocoon.properties (aus Verzeichnis *cocoon\src\org\apache\cocoon*) nach *Jrun\properties*

Servlet-Engine konfigurieren:
Jrun Administrator über über Programme\JrunAdministrator starten
jsm-default auswählen, configure anklicken
jseweb wählen, service configuration anklicken
auf Registerkarte Aliases folgendes hinzufügen:
bei Name: **cocoon**
bei Class name: **org.apache.cocoon**
bei Init Args: **Jrun\properties\cocoon.properties**

Auf Registerkarte Mapping folgendes hinzufügen:
bei Virtual Path Extension: ***.xml**
bei Servlet Invoked: **cocoon**
Alles abspeichern!

Test:
Falls Sie nun etwas testen wollen, müssen Sie Ihre Testdateien nach *Jrun\jsm-default\services\jws\htdocs* kopieren. Wenn Ihnen das zu umständlich ist, können Sie den Pfadeintrag in der Service-Configuration des jws-Servers ändern.

Jetzt können Sie über Programme\Jrun\StartJRun die Servlet Engine starten, den Browser starten und vom jws-Webserver das gewünschte Dokument anfordern, z.B.:
http://localhost:8000/probe.xml
Das Servlet gibt Ihnen eine HTML Seite aus. Wenn Sie sich das Ergebnis im Quellcode ansehen, kommt unten eine Statusmeldung, daß Cocoon das Dokument verarbeitet hat und wie lange es dafür gebraucht hat.

Tuning:
Leider braucht die Verarbeitung mit Jrun unter NT wesentlich länger als bei Jserv und Apache. Man kann jedoch die Java Hotspot Engine von Sun benutzen, die das Ganze erheblich beschleunigt und sehr leicht zu integrieren ist (derzeit nur Windows NT/98/95).
- Hotspot Download von: javasoft.com, ca. 1,5 MB
- Verzeichnis in JRUN\jre\bin\ anlegen und files entpacken

6.2 eXcelon

Excelon ist ein kommerzielles Client-Serversystem, das die Speicherung von allen Datenformaten und die Verwaltung von XML Daten unterstützt. Die Herstellerfirma ObjectDesign positioniert das System jedoch weniger im Bereich content management als im Bereich XML-Entwicklungsumgebung.

Serverseitig wird eine ObjectStore-Server mit Cache-Manager (das bedeutet im Hintergrund arbeitet eine objektorientierte Datenbank), Java-Klassen und einige DLLs zur Systemverwaltung mit der MS System Management Console ausgeliefert. Letzteres bedeutet: der Server läuft zur Zeit nur auf Windows NT. Mit Version 1.1 soll zum August 1999 auch eine Solaris - Version verfügbar sein.
In der Datenbank können prinzipiell alle Daten gespeichert werden. Alle MIME Typen können per drag and drop nach eXcelon importiert und exportiert werden. (In der Version 1.0 sollte man tunlichst drag and drop verwenden, denn der Import via Menübefehlen führte dazu, daß xsl Dateien nicht als xml erkannt wurden Dieser

bestätigte Bug soll laut eXcelon Diskussionsliste mit Version 1.1 behoben sein) Mit den zugehörigen Client-Programmen können lediglich XML-Daten manipuliert werden. Der Server (cache-manager) verwaltet Pufferspeicherbereiche, die prinzipiell auch auf mehrere Server-Maschinen verteilt sein können.

Bei der Installation wird eXcelon 1.0 grundsätzlich unter dem System Account eingerichtet. Der Server-Service wird auch unter diesem account gestartet, egal unter welcher Berechtigung der Installateur angemeldet war. Dies führt dazu, daß ein NT Benutzer, auch mit administrativen Rechten, keine Berechtigung hat, einen Connect auf den Server durchzuführen. Die Einstellung der Startart in Einstellungen/Systemsteuerung/Dienste auf einen spezifischen user-account kann hierbei Abhilfe schaffen.

Clientprogramme sind die Programme *Manager*, *Explorer*, *Studio*. Auch der Webzugriff, der über den Microsoft Internet Information Server oder über einen Netscape Server möglich ist, ist eine Client-Anwendung in Form der xlnisapi.dll. Alle Client Programme kommunizieren mit den Funktionen der XML Datenbank über das eXcelon client API (CALL Level API zur xml DB). Dieses API kann auch genutzt werden, um eigene Client Programme zu erstellen.

6.2.1 eXcelon Manager

eXcelon Manager ist ein Verwaltungsprogramm, das es ermöglicht, XML Speicherbereiche (XMLStores) und Verzeichnisstrukturen anzulegen, Server-erweiterungen zu installieren und Benutzer- und Zugriffsberechtigungen zu verwalten. Darüberhinaus kann mit dem Manager Einfluß auf Caches und deren Verhalten genommen werden. Der Manager ist als sog. snap-in in die Microsoft Management Console integriert, setzt also einen installierten Option Pack von WinNT voraus.

6.2.2 eXcelon Explorer

Explorer ist das Datenverwaltungsprogramm des Systems. Auch Explorer kann verwendet werden, um Verzeichnisse anzulegen. Die Darstellung der Datenbankinhalte entspricht der Baumdarstellung des Windows-Dateisystems. Daten können aus dem Dateisystem oder aus ODBC / OLE DB Datenquellen importiert werden.
Der Import aus unseren eigenen relationalen Datenbeständen via ODBC mißlang jedoch. Die erzeugten XML-Tags enthielten mitunter nicht die Feldnamen der importierten Tabellen und Abfragen, sondern z.T. deren Inhalte. Ein spezieller Treiber von Microsoft aus dem Data Access pack, soll angeblich bessere Ergebnisse liefern.

Sind XML-Daten im System integriert, können auf diesen Daten Abfragen ausgeführt werden. Excelon bietet hierzu eine Abfragesprache, die sich an den Vorschlägen und Beschränkungen des W3C zu einer XML Query Language (XQL) orientiert.
Wahlweise können Abfragen manuell, oder - mit eingeschränktem Umfang - mit einem Query Generator per Mausklick erstellt werden. Die Abfragen stellen derzeit sicherlich den größten Schwachpunkt des Systems dar. Es ist nicht möglich, Abfragen zu definieren, die reguläre Ausdrücke auswerten. So können zwar mit

```
/ADRESSLISTE/PERSON[NACHNAME = "Knobloch"]
```

die Einträge zur Adresse gefunden werden, Konstrukte wie

```
/ADRESSLISTE/PERSON[NACHNAME = "K*"]
```

können derzeit aber noch nicht verarbietet werden; d.h. die Suchzeichenfolge muß exakt dem Eintrag entsprechen. Laut Aussagen der Entwickler, wird diese Fähigkeit noch im Lauf des Jahres eingebaut.
Ebenfalls existiert keine Möglichkeit, Ergebnismengen von Abfragen zu sortieren. Eine Sortierung muß derzeit über xsl-Anweisungen in der Repräsentation gemacht werden. Eine Abfrage kann sich außerdem nur auf einen XML-Datenbestand beziehen, was eine weitere Einschränkung ist.

Inhalte können sowohl in einer XML-Strukturanzeige, als auch in einem XML-fähigen Browser angezeigt werden. Per Mausklick ist es möglich, zwischen den Anzeigefenstern und damit zwischen den Darstellungsformen hin- und herzuschalten. Deutsche Umlaute werden jedoch nur angezeigt, wenn ie5 als Browser aktiv ist. Version 1.1 verspricht als Hauptverbesserung hier Abhilfe mit der Unterstützung der gängigen internationalen Zeichensätze.

Ebenfalls möglich ist es, XML-Dateien zu editieren. Des Edit-Werkzeug generiert dabei Statements in der XML-Update-Sprache und führt diese aus. Das update-Fenster (update window) erlaubt das Ändern von Inhalten wie auch das Ändern der Struktur, d.h. es können Elemente, Attribute und Kommentare eingefügt werden. Alle Daten, die nicht explizit als xml-Dateien bekannt sind, können nicht im Explorer bearbeitet werden.

6.2.3 eXcelon Studio

Studio ist ein grafisches Werkzeug, mit dem XML Strukturen und Relationen zwischen diesen angelegt werden können. Die Strukturen, die erzeugt werden, legt eXcelon Studio im Document Content Description (DCD) Format ab und generiert daraus Datenbankschemata. Mit einem Menübefehl kann Studio aus diesen Schemata entsprechende Java-Programme generieren, die die notwendigen *add*, *edit* und *delete* Methoden realisieren. Mitgeneriert werden auch bat-Dateien, die für *compile* und *installation* dieser objektbezogenen Methoden im eXcelon-Server sorgen. Ein weiterer Aufruf des Menüpunkts generiert HTML-Formulare und ASP-Scripts, die einen webbasierten Zugriff auf die Servererweiterungen implementieren. Auf diese Weise ist es möglich, in kürzester Zeit einen XML-Datenspeicher mit webbasierter Verwaltung zu generieren.

6.2.4 Die WebServer Erweiterungen

Ebenfalls als Client-Anwendung funktioniert die WebServer Erweiterung, die es ermöglicht, sofort nach der Installation von EXcelon über das WWW auf die Daten des XML Speichers zuzugreifen. Dazu wird die mitgelieferte xlnisapi.dll wie ein CGI-Script aufgerufen und entsprechend parametrisiert.

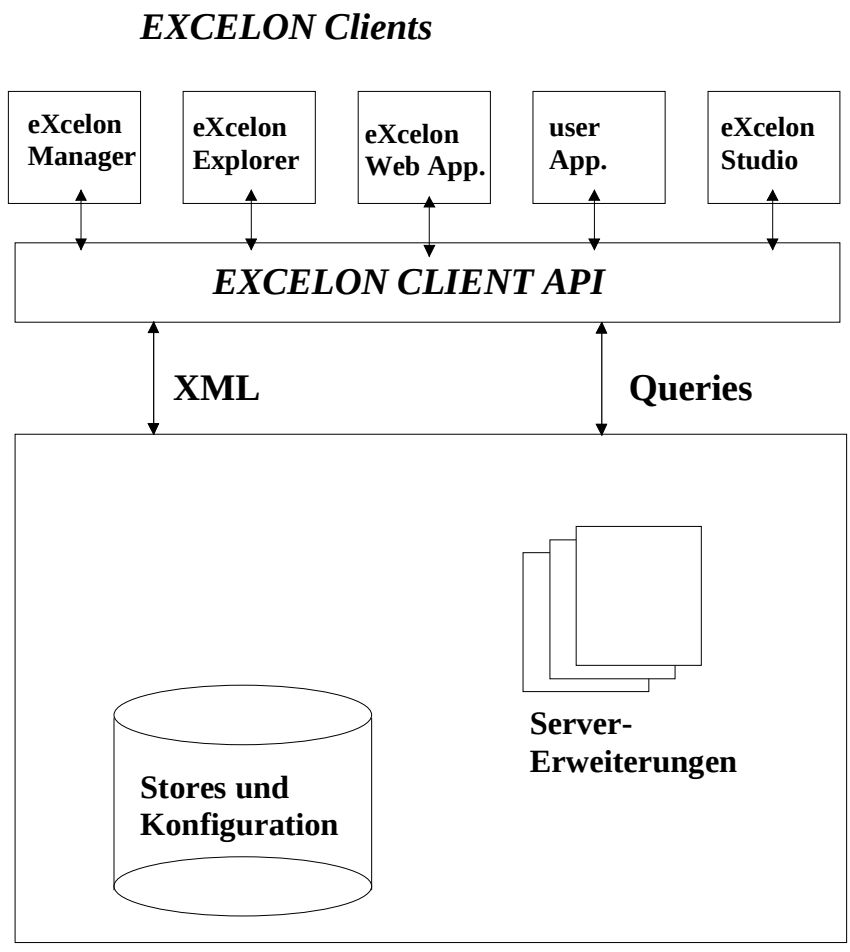
http://herakles.diff.uni-tuebingen.de/Scripts/xlnisapi.dll/KMMTDB/Adresse_von_knobi.xql

öffnet die xml Abfrage "Adresse_von_knobi.xql" auf dem Server und gibt das Ergebinis als XML-Datenstrom aus.

Mit dieser Methode können Dokumente, gespeicherte Abfragen und selbsterstellte Servererweiterungen über eine Referenz aktiviert werden. Dies kann ebenso über HTML-Formulare geschehen.
Neben der Unterstützung von MS Internet Information Server und den Netscape Servern ist für die Version 1.1 die Unterstützung der Apache Web-Servers angekündigt.

Server Erweiterungen:

Erweiterungen zur Funktionalität des eXcelon-Servers können durch selbst geschriebene Programme realisiert werden. Wird Java als Sprache benutzt muß das Programm bestimmte mitgelieferte Server-Klassen importieren und ein oder mehrere Interfaces implementieren. Nach erfolgreichem *compile* kann das *classfile* installiert werden. Clients aktivieren eine selbsterstellte Server-Erweiterung durch eine Datei, die eine Referenz auf die Server-Erweiterung darstellt. Auch diese muß vom Entwickler angelegt werden. Bei der Arbeit mit Studio wird der Code für Server Erweiterungen derzeit in Java generiert. Bei der Eigenentwicklung von Server-Erweiterungen kann prinzipiell jede Programmiersprache verwendet werden, die COM-Objekte unterstützt. Also auch eingefleischte C++ oder VB Programmierer können sich betätigen.



7 Anhang

7.1 Syntax für XSL Patterns (Suchmuster)

XML Dokumente repräsentieren eine Hierarchie von Knoten, ähnlich der Hierarchie in einer Verzeichnisstruktur. Diese Ähnlichkeit ist auch der Grund für die Ähnlichkeit zwischen URS und XSL Mustern.
XSL Muster können jeden Knoten des XSL Dokumentes als Ausgangspunkt (context) für eine Suche annehmen. In einem XSL Style Sheet ist der Kontext für eine Suche der Knoten, der gegenwärtig durch ein **<xsl:template>** oder ein **<xsl:for-each>** Element verarbeitet wird. Sehr häufig wird als Basis der Kontext des **root-node** (**Wurzelknoten** des XML Dokumentes) verwendet.
Auch unsere Beispiele beginnen beim **root node**.

```
EXPERTENNETZ/EXPERTE/ADRESSE

Ein einfaches XSL Muster beschreibt einen Pfad durch eine XML Hierarchie mit
einer Liste von Kindelementen, die durch Schrägstriche getrennt werden.
Das Muster identifiziert alle Elemente die Kinder von Adresse sind.
```

Man kann auch `wildcards` verwenden, um unbekannte Elemente zu beschreiben. Ein Element wird durch "*" repräsentiert.

```
EXPERTENNETZ/*/ADRESSE

Die obige Suche würde alle Elemente Adresse suchen, unabhängig davon,
welches Parent-Element (Elternelement) diese haben.
```

```
EXPERTENNETZ/*

Dieses Muster sucht alle Kindelemente von Expertennetz, egal wie sie heißen
```

```
EXPERTENNETZ/EXPERTE/ADRESSE[EMAIL]/NAME

Man kann auch Verzeigungen mit eckigem Klammern angeben. Die folgende Suche
verzeigt ins Element ADRESSE, sucht aber nur die, die als Kindelement ein
EMAIL Element haben: Man kann in den eckigen Klammern auch einen String als
Vergleich angeben: [EMAIL=‘ ... ‘]
```

```
EXPERTENNETZ/EXPERTE/ADRESSE/INSTITUTION/@TYP

Attribute werden mit einem Klammeraffen @ angegeben.
Gesucht wird ein Element INSTITUTION mit dem Attribut TYP, das den Wert
Projekt des Landes BaWue hat.
EXPERTENNETZ/EXPERTE/ADRESSE/INSTITUTION[@TYP="Projekt des Landes BaWue"]
```

7.2 Links

XML-Spezifikationen:

http://www.w3.org/TR/1998/REC-xml-19980210
Das normative englische Original, nach dem man sich im Zweifelsfall richten sollte.

http://www.oasis-open.org/cover/xml.html#reference
Die XML-Spezifikation in allen möglichen Sprachen und Dateiformaten

http://www.mintert.com/xml/trans/REC-xml-19980210-de.html
Die deutsche Fassung der Spezifikation (ist auch im Buch von Mintert/Behme, s. Literatur)

XSL-Spezifikation:

http://www.w3.org/TR/WD-xslt
Der Working Draft zum Teil I, der XSL Transformation Language

http://www.w3.org/TR/WD-xsl/
Der Working Draft zum Teil II, der XSL Formatierungs Language

XML-Softwaresammlungen:

http://www.xmlsoftware.com
Vorbildlich kategorisierte und kommentierte Sammlung von XML und verwandten Technologien, die regelmäßig gepflegt wird.

http://www.alphaWorks.ibm.com/Home/
Die neuen Technologien und Tools, die IBM auf der Basis von XML bzw. XML und Java anbietet.

XML-Anwendungen und -projekte:
http://www.oasis-open.org/cover/xml.html#applications
Kurzbeschreibungen und – sofern vorhanden – Links zu bereits realisierten oder in Arbeit befindlichen Software-Lösungen mit XML

http://msdn.microsoft.com/xml/scenario/intro.asp
Verschiedene Szenarios zum webbasierten Einsatz von XML.

http://java.apache.org/
Uner der Aegide von Stefano Marzocchi läuft hier das Cocoon Projekt. Es ist generiert serverseitig automatisch HTML aus XML/XSL, wird aber in Zukunft noch wesentlich mehr können.

XML/XSL-Tutorials und Beispiele zum Lernen:

Generell noch nichts allumfassendes, aber das kann sich ja noch ändern:

http://members.aol.com/xmldoku
Eine kurze nichttechnische Einführung für echte Anfänger, die einen ersten Überblick gibt. Optisch nicht gerade berauschend, aber ganz brauchbar, um eine Vorstellung für Sinn und Zweck von XML zu bekommen.

http://www.stud.ifi.uio.no/~larsga/download/artikler/processing.pdf
PDF-Folien zum Download. Sehr guter Überblick, der in die Interna hauptsächlich von XML Prozessoren und konzeptuelle Probleme einführt.

http://msdn.microsoft.com/xml/tutorial/default.asp
Interaktiver Workshop mit mehreren Lektionen. Die Lernkurve steigt nach den Einführungslektionen allerdings ziemlich steil an. Das Laden der Asp-Seiten dauert unerträglich lang.

http://msdn.microsoft.com/xml/xmlguide/default.asp
Gehaltvoller Workshop für XML Entwickler mit guter Inhaltsübersicht. Das Laden der Asp-Seiten dauert und dauert und dauert ....

http://metalab.unc.edu/xml/books/bible/examples/
Eine Menge XML/XSL/CSS-Dateien zur sogenannten XML Bible von Elliotte Rusty Harold (s. Literaturliste). Wohl nur mit Buch oder Vorkenntnissen sinnvoll.

XML bzw. XSL mailing-Listen und newsgroups

http://www.mulberrytech.com/xsl/xsl-list
Liste mit regem Verkehr zu Themen sehr unterschiedlicher Richtung. Auch die Profis diskutieren heftig mit.

comp.infosystems.www.authoring.stylesheets

XML- Listen und Links zu verschiedenen Themen:

Deutschland:
http://www.fzi.de/v1/vfw/xml/home.html
Eine Reihe von Powerpoint-Folien zum Thema XML vom Forschungszentrum Informatik, Karslruhe sowie eine kleine Auswahl von Links. Einige davon sind sonst kaum zu finden, aber lohnenswert.

http://www.heise.de/ix/raven/Web/xml/default.html
Der Heise Verlag unterhält im Rahmen der Zeitschrift iX eine XML-Site mit vielen nützlichen Links zu Projekte, Konferenzen und FAQs. Sehr gepflegt und mit leichtem Zugang zu den aktuellsten Entwicklung der ‚Working Papers'.

USA:
http://www.WDVL.com/Authoring/Languages/XML/Resources.html
Riesige Sammlung zu so ziemlich allen Themen rund um XML. Die Sammlung ist bisher gut gepflegt.

http://metalab.unc.edu/xml/
Elliotte Rusty Harold's Cafe con Leche Site bietet so ziemlich alles und immer das Neueste zum Thema XML. Wichtig sind vor allem Konferenzen und News. Das XML-Buch von E. Harold ist auch auf der Literaturliste.

http://www.xml.com
XML.com ist ein Partnerprojekt von Seybold Publications und Songline Studios, das zu O'Reilly & Associates gehört. Es bietet oftmals ausgezeichnete Artikel von den ‚Mitautoren' von XML wie z.B. Tim Bray, Lisa Rein, Norman Walsh oder James Clark.

Hilfsmittel
http://www.csclub.uwaterloo.ca/u/mvcorks/code/character-set.html
Liste der ISO 8859-1 and 10646 Zeichen als SGML Entities

7.3 Literatur

Vorbemerkung:
Literatur zu XML gibt es wie Sand am Meer. Da man XML recht schnell gelernt hat und XSL noch kein fertiger Standard ist, sollten Sie sich im Moment nicht zu viele Bücher anschaffen, sondern lieber die Online-Quellen nutzen. Wir haben hier bewußt nur eine kleine Auswahl getroffen.

Presenting XML: *Light, Richard;* 1997, ISBN 1-57521-334-6, DM 42.-

Structuring XML Documents: *Megginson, David;*
1997, ISBN 0-13-642299-3, DM 88.-

XML Handbook: Goldfarb, Charles F.; Prescod, Paul; 1998 dt. 1999, ISBN 0-13-081152-1, Prentice Hall, Imprint von Markt &Technik, DM 99.-

XML in der Praxis: Professionelles Web-Publishing mit der Extensible Markup Language; Behme, Henning, Mintert, Stefan, 1998, Addison-Wesley-Longman, ISBN 3-8273-1330-9, DM 69,90.

XML, Extensible Markup Language: Elliotte Rusty Harold, 1998, IDG Books, ISBN 0-7645-3199-9, $ 39.99.